A Model-Based Security Testing Approach for Automotive Over-The-Air Updates

Mahmood, S, Fouillade, A, Nguyen, HN & Shaikh, S

DOI 10.1109/ICSTW50294.2020.00019

Publisher: IEEE

A Model-Based Security Testing Approach for Automotive Over-The-Air Updates

Shahid Mahmood*† Alexy Fouillade‡ Hoang Nga Nguyen*§ Siraj A. Shaikh*¶
* Systems Security Group, Institute for Future Transport and Cities, Coventry University, Coventry, United Kingdom
‡École supérieure d'électronique de l'Ouest, Grande Ecole d'Ingénieurs Généralistes à Angers, Angers, France
†mahmo136@uni.coventry.ac.uk, ‡alexy.fouillade@reseau.eseo.fr
§ac1222@coventry.ac.uk, ¶aa8135@coventry.ac.uk

***Abstract*—Modern connected cars are exposed to various cybersecurity threats due to the sophisticated computing and connectivity technologies they host for providing enhanced user experience for their occupants by offering numerous innovative applications. While prior studies exist that explore cybersecurity challenges, tools and techniques for automotive systems, over-the-air (OTA) software updates for automobiles can be exploited by the attackers to compromise vehicle security and safety has not been covered extensively. This paper presents our Model-Based Security Testing (MBST) approach, designed for cybersecurity evaluation of the OTA update system for automobiles, which has an integrated testbed and a software tool that is capable of automatically generating and executing test cases by using attack trees as an input. Integrating threat modelling in the testing provides several benefits, including clear and systematic identification of different threats. Automation of the test-case generation and execution has the obvious benefits of saving time and manual effort, as manual test-case generation is both a time-consuming and error-prone process (especially, when the testing involves several test-cases). A simple simulated attack is used to demonstrate the validity and effectiveness of our testing approach. To the best of our knowledge, there is no prior research that uses a testing approach similar to our approach for automotive OTA security evaluation.**

***Keywords*-over-the-air updates, OTA, automotive, cybersecurity, testing, testbed, testing approach, model-based security testing, attack tree**

I. Introduction

Modern vehicles are increasingly vulnerable to cybersecurity attacks due to the embedded computing and internet connectivity capabilities they are equipped with. As cyberattacks have the potential to seriously undermine the safety of an automobile and its occupants, effective testing for detecting software flaws and weaknesses is crucial. The software and firmware installed on these in-vehicle embedded computing devices need regular updates, carrying critical security patches, bug fixes, and other enhancements for improved functionality. Previously, installation of these updates required the vehicles to visit the dealership's service centres (which is still the case for updating the firmware of safety-critical components including braking, steering and engine control etc.), via onboard diagnostic ports. OTA software update system has emerged as a convenient, efficient, and cost-effective alternative for delivering updates to automobiles remotely, which offers many benefits including significantly reduced costs and opportunity for continuous, seamless improvement. While there are several advantages of OTA updates, security threats that they introduce must also be considered seriously. While existing literature on cybersecurity testing of automotive systems extensively explores security challenges, relevant solutions, testing techniques and testing environments focusing on various attack surfaces and vectors, security testing of the OTA for automobiles has not been considered adequately. In fact, very few studies have been published in this important area. In this paper, we present our approach for evaluating OTA cybersecurity that relies on a software tool for automatic test-case generation and execution. Our approach uses attack trees for threat modelling which provide numerous benefits including a clear understanding of different potential attacks from the attacker's perspective. Our software tool uses attack trees for automatic test-case generation, which is a highly useful feature in terms of saving time and manual efforts in the testing process. This is particularly relevant and useful when the testing has a large number of test cases.

We also present details of a simulated attack performed against a reference implementation of the Uptane Framework [20], which is an OTA software update system, specially designed for automotive systems. Our experiment attack involves compromising Uptane repositories hosting firmware image files and associated metadata. In addition to this, an overview of our testbed, the tool for automated test-case generation and execution, and testing approach has been provided. Our main contributions include the following:

- cybersecurity testing of automotive OTA updates using a systematic, model-based security testing approach
- automation of test-case generation and execution using attack trees
- a testbed for automotive cybersecurity testing

To the best of our knowledge, there is no prior study that uses a model-based approach for security testing of automotive OTA updates.

The rest of the paper is organised as follows: Related work is presented in Section II, Section III provides background information about automotive security testing, Uptane framework, attack trees, Communication Sequential Processes (CSP), and our integrated test-case generation and execution tool. Section IV gives an overview of our testing approach by describing its four key stages, followed by Section V that presents

details of our testbed for OTA security evaluation. Details of the simulated attack aiming at compromising the OTA repositories are presented in Section VI, which is followed by the conclusion in section VII.

II. Related Work

Model-based security testing is concerned with specifying, documenting and generating security test objectives, test cases, and test suites in a systematic and efficient manner [30].

Santos et al. [29] propose their automotive cybersecurity testing framework, which uses CSP for representing the models of the vehicle's bus systems as well as a set of attacks against these systems. CSP - a language with its own syntax and semantics - is a process-algebraic formalism used to model and analyze concurrent systems. Using CSP, they create architectures of the vehicle's network and bus systems along with the attack models. One of the key challenges that authors claim to address in their work is the scalability of the testing in distributed environments. Their system model is comprised of networks, bus systems connected to each network, and the gateways. Additionally, network parameters, such as latency can also be modelled. An attack model is also created, defining the attackers' capabilities as channels. An attacker's capabilities may include command spoofing, communication disruption, eavesdropping and influencing behaviours of the system. According to the authors, the ability for a detailed definition of the scope of the attack and test cases is a key advantage of using these models for security testing.

Wasicek et al. [23] present aspect-oriented modelling (AOM) as a powerful technique for security evaluation of Cyber-Physical Systems (CPS), especially focusing on safety-critical elements in automotive control systems. AOM is based on the ideas inspired by aspect-oriented programming, which is concerned with crosscutting aspects being expressed as concerns (e.g., security, quality of service, caching etc.) [9]. Aspect-oriented modelling is used to express crosscutting concerns at a higher level of abstraction by means of modelling elements [5]. The technique presented by [23] models attacks as aspects, and aims at discovering and fixing potential security flaws and vulnerabilities at design time, because it becomes highly costly to find and fix the bugs if they are discovered later in the development life-cycle stages for automotive systems. Some of the main benefits that can be achieved by using AOM for security assessment of automotive systems include: separation of functional and attack models into aspects allows domain experts to work on different aspects without any interference; real-world attack scenarios involving high degree of risks can be modelled easily; general models can be reused in other systems. An automotive case study is presented by the authors, involving the adaptive cruise control system as an example. They use a special modelling and simulation framework, called Ptolemy II, for developing their models. The authors intended to explore the effects of attacks on the communication between two vehicles. A discussion of four different attacks (i.e., man-in-the-middle, fuzzing, interruption, and replay) is presented.

III. Background

A. Automotive security testing

Modern automobiles are exposed to numerous cybersecurity threats due to their builtin powerful computing and communication capabilities. Identifying vulnerabilities and security flaws in the communication and other onboard technologies in connected cars is critical, as cybercriminals can exploit those weaknesses for gaining access to the safety-critical systems of the vehicle.

Most cars today host many computing devices, known as Electronic Control Units (ECUs). Each ECU has specific responsibilities, and they may need to communicate with each other and with the external world for successful completion of their tasks. For local communication, they rely on one or more of the in-vehicle communication networks, such as Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, and Media-Oriented Systems Transport (MOST). Each type of network has been designed to support applications with different needs. For example, while LIN is mostly used for low-speed applications, applications requiring high-speed data-transfers use MOST [14]. Legally mandated Onboard Diagnostic (OBD) ports in the modern vehicles are used for ECU firmware updates, repairing and inspections of the vehicle. They are also used for reporting the data gathered by various sensors in the car to the outside world, providing information on the health status of the vehicle [31].

There are several entry points that attackers can take advantage of for breaking into a vehicle's internal system, which have been extensively explored and presented by previous studies. For example, [4], [16], [27] explore CAN exploitation, [24] reports attacks leveraging OBD port, and security issues related to in-vehicle infotainment are presented in [18]. Over-the-air (OTA) software update systems for automobiles can also be targeted by hackers in several different ways, as described in [20] for compromising the security and safety of the connected vehicles.

While automotive OTA offers numerous benefits (e.g., seamless delivery of software updates remotely), presence of security flaws and vulnerabilities in such systems can be exploited by adversaries to undermine the security of connected cars. For example, attackers can compromise the repositories that host the software updates, as described by Kuppusamy et al. in [20]. Various testing methods (for example, [2], [3], [6]–[8], [11], [12], [15], [22], [29]) and testing environments (e.g., [10], [13], [32], [35], [36]) have been proposed for the security testing of automotive systems. These testbeds and techniques have been designed primarily for discovering security flaws in vehicular networks (e.g., CAN, MOST, LIN, etc.), ECUs, and IVIs. Cybersecurity testing of the automotive OTA software update systems has not been considered by these works.

B. The Uptane framework

Uptane, developed in collaboration with automotive industry stakeholders in the US, is an automotive software update framework, which is claimed to address automotive-specific

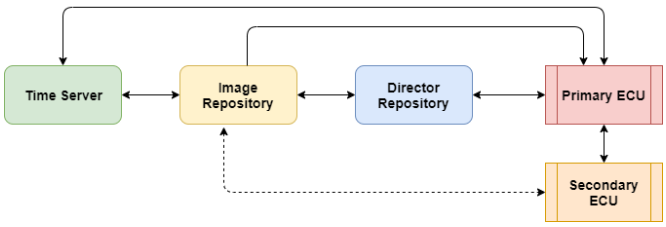

Fig. 1. An overview of the Uptane framework, illustrating the interconnections and flow of information among the Time Server, Image Repository, Director Repository, Primary ECU and Secondary ECU.

security flaws, and provide protection against a wide range of security attacks.

As shown in Figure 1, Uptane framework has three key components: the Image Repository, the Director Repository, and the Time Server. The Image repository holds all the images deployed by the OEM along with metadata files for proving the authenticity of the hosted images. The Director Repository is responsible for tracking and determining what update to be delivered to each ECU based on the current status of the repository. As time is a critical aspect in automotive software updates, knowledge of current, accurate time is crucial for the vehicle. Many ECUs are unaware of current time because they do not have clocks, this is where the Time Server plays an important role in providing current time to the vehicle in a cryptographically secure manner. More comprehensive introduction of the framework can be found in [33].

A primary ECU is typically the one that is more capable in terms of storage capacity and connectivity as compared to a secondary ECU which needs help from the primary ECU for receiving and installing software updates.

C. Attack trees

Attack trees contain a goal (the root of the tree), a set of sub-goals, structured using the operators conjunction (**AND**) and disjunction (**OR**), and leaf nodes, which represent atomic attacker actions. The **AND** nodes are complete when all child nodes are carried out and the **OR** nodes are complete when at least one child node is complete.

Extensions have been proposed using **Sequential AND** (or **SAND**) [17]. We follow the formalisation of attack trees given in [17], [21]. If $\mathbb{A}$ is the set of possible atomic attacker actions, the elements of the attack tree $\mathbb{T}$ are $\mathbb{A} \cup \{\mathbf{OR}, \mathbf{AND}, \mathbf{SAND}\}$, and an attack tree is generated by the following grammar, where $a \in \mathbb{A}$:

$$t ::= a \mid \mathbf{OR}(t, \ldots, t) \mid \mathbf{AND}(t, \ldots, t) \mid \mathbf{SAND}(t, \ldots, t)$$

Attack tree semantics have been defined by interpreting the attack tree as a set of series-parallel (SP) graphs [17].

D. Communicating Sequential Processes (CSP)

We give here a brief overview of the subset of CSP used in this study. A more complete introduction may be found in [28]. Given a set of events Σ, CSP processes are defined by the following syntax:

$$P \quad ::= \quad \mathit{Stop} \mid e \rightarrow P \mid P_1 \,\Box\, P_2 \mid P_1;\ P_2 \mid P_1 \underset{A}{\parallel} P_2 \mid P_1 \,|||\, P_2$$

where $e \in \Sigma$ and $A \subseteq \Sigma$. For convenience, the set of CSP processes defined via the above syntax is denoted by CSP. To mark the termination of a process, a special event $\checkmark$ is used. In the above definition, the process *Stop* is the most basic one, which does not engage in any event and represents deadlock. In addition, *Skip* is an abbreviation for $\checkmark \rightarrow \mathit{Stop}$. It only exhibits $\checkmark$ and then behaves as *Stop*. The prefix $e \rightarrow P$ specifies a process that is only willing to engage in the event e, then behaves as P. The external choice $P_1 \,\Box\, P_2$ behaves either as P_1 or as P_2. The sequential composition $P_1;\ P_2$ initially behaves as P_1 until P_1 terminates, then continues as P_2. The generalised parallel operator $P_1 \underset{A}{\parallel} P_2$ requires P_1 and P_2 to synchronise on events in $A \cup \{\checkmark\}$. All other events are executed independently. Finally, the interleaving operator $P_1 \,|||\, P_2$ allows both P_1 and P_2 to execute concurrently and independently, except for $\checkmark$.

There are different semantics models for CSP processes, for further detail please refer to [28].

IV. MBST Approach

Systematic cybersecurity evaluation of automotive systems is a non-trivial, critical task. Comprehensive security assessment requires a disciplined and well thought out approach. As opposed to an ad-hoc testing approach which often suffers from subjective prioritization of test cases leaving numerous undiscovered vulnerabilities in the system, a methodical approach increases the chances of detecting more flaws.

In this section, we present details of our testing approach[1] that incorporates a software tool for generating and executing test cases automatically. A testbed is also a part of our approach which is described in the next section.

Our testing approach is inspired by the Penetration Testing and Execution Standard (PTES) [26] and model-based security testing [8]. A key feature of the PTES testing methodology is the use of threat modelling techniques. We use attack trees for threat modelling since they support and facilitate our automated test case generation and execution process. Our approach, comprising four different stages, is depicted in Figure 2. In general, specifications and implementation details of the automotive systems are not accessible due to commercial sensitivity and obscurity of subsystems; therefore, reconnaissance or intelligence gathering must be performed in order to discover potential vulnerabilities in the target system. Our approach is primarily concerned with revealing potential known flaws and undesirable behaviour of the system by looking at it from the perspective of an attacker.

An overview of the stages of our approach is presented in the following subsections.

[1]The source code for the test-case generation/execution tool and Uptane reference implementation along with a guide for setting up all components can be downloaded from https://tinyurl.com/rydjmqa.

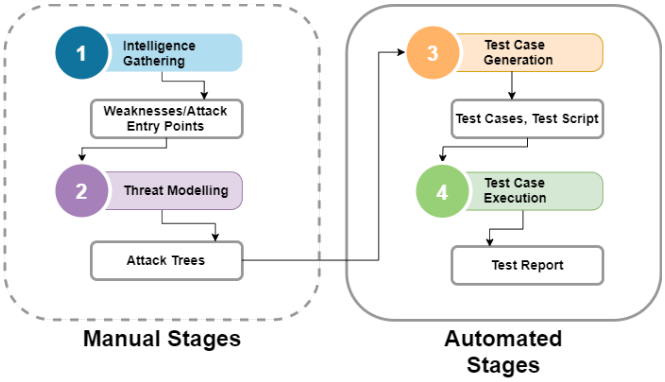

Fig. 2. An overview of our testing approach for automotive OTA update system, showing key stages, inputs and outputs of each stage. First two stages involve activities that are performed manually. Test case generation and test case execution are fully automated.

A. Intelligence gathering

This stage involves gathering as much information about the target system as possible, particularly, any known weaknesses and exposed attack surfaces. This may also include looking into published documentation as well as the specification or source code of the system if available.

B. Threat modelling

Once enough information has been collected, the attack trees can be created using the ADTool, a threat modelling and analysis tool developed by researchers from the University of Luxembourg [19]. Attack trees are used for the identification of various potential threats to a system from the perspective of an attacker. Being a structured approach, attack trees enable systematic security evaluation by focusing on threats and associated actions that can be performed by the attacker for launching attacks.

C. Test case generation

Security test cases are generated by model-checking, which is a model-based technique. It is worth noting that we modeled the potential threats by using attack trees, we did not create a model of the system under test (SUT).

The test-case generation process begins with creating an attack tree that provides the basis for subsequent steps in the process. Attack tree creation requires clear identification of the attack goal and possible relevant actions that can be performed to launch the attack.

An initial prototype of our test-case generation tool was first introduced in [6], which has been adapted for the current study. The alterations made to the software tool include some enhancements related to input and output. Changes to the input system have been made to allow the tool to accept XML-based attack tree files. Similarly, essential amendments have been applied in order to ensure that the generated test scripts are compatible with the target system (i.e. the system under test).

Test cases are automatically generated using FDR, the refinement checker for CSP. To this end, attack trees must be first translated into CSP processes. In principle, the logic gates of the attack tree can be considered CSP operators [28] as follows:

- Since the AND logic gate demands that all actions must be successful for the branch to be considered complete, the interleave operator ($|||$) is used. This operator joins processes that operate concurrently but without them necessarily interacting or synchronising.
- The sequential composition operator (;) is used for the SAND logic gate. The former echoes the SAND logic gate, in that the first process must terminate successfully before the next is allowed;
- The external choice operator ($\Box$) (where any process could be chosen dependent on the environment in which it operates) is used for the OR logic gate.

Formally, we define the following transformation function $\mathrm{trans} : \mathbb{T}_{SAND} \to CSP$ where $\Sigma = \mathbb{A}$:

- $\mathrm{trans}(a) = a \to Skip$ for $a \in \mathbb{A}$;
- $\mathrm{trans}(OR(t_1, \ldots, t_n)) = \mathrm{trans}(t_1) \,\Box\, \ldots \,\Box\, \mathrm{trans}(t_n)$;
- $\mathrm{trans}(AND(t_1, \ldots, t_n)) = \mathrm{trans}(t_1) \,|||\, \ldots \,|||\, \mathrm{trans}(t_n)$;
- $\mathrm{trans}(SAND(t_1, \ldots, t_n)) = \mathrm{trans}(t_1);\ \ldots;\ \mathrm{trans}(t_n)$;

Once $\mathrm{trans}(t)$ is obtained, trace refinement is used to extract test cases following [25]. To this end, $\mathrm{trans}(t)$ acts as a filter criterion to select test cases among all possible runs of the system captured by *Sys*. As in [25], we define a fresh event *attackSucceed* to mark the end of an attack, which indicates that an attack is successfully executed. We form the following filter

$$TestPurpose = \mathrm{trans}(t);\ (attackSucceed \to Stop)$$

which captures all attacks extended with the marking event *attackSucceed* at the end. Then, we establish the following trace refinement:

$$Sys \,\Box\, TestCases \sqsubseteq_T Sys \underset{\Sigma \setminus \{attackSucceed\}}{\|} TestPurpose$$

In this refinement, *TestCases* encodes test cases that have previously been generated. By combining it with *Sys* using the external choice operator, a fresh test case, i.e., different from the generated ones, will be generated if one exists. $Sys \underset{\Sigma \setminus \{attackSucceed\}}{\|} TestPurpose$ encapsulates all attack traces that can be carried out with respect to the formal model *Sys*. These attack traces are ended with the marking event *attackSucceed*, which does not belong to *Sys*, hence, gives rise to counterexamples of the refinement. Initially, $TestCases = TestCases_0 = Stop$, i.e., corresponding to an empty set of test cases. This refinement is checked by calling FDR [1]. If an attack trace exists, FDR provides a counter example of the form $\langle a_1, \ldots, a_n, attackSucceed \rangle$ where $a_1, \ldots, a_n \in \Sigma \setminus \{attackSucceed\}$. We encode this trace as a test case $tc_1 = a_1 \to \ldots \to a_n \to attackSucceed \to Stop$. After *TestCases* is rebuilt as $TestCases = TestCases_1 = TestCases_0 \,\Box\, tc_1$, the above refinement check is called again and again to extract further test cases $tc_2, \ldots$ and to construct $TestCase_2, \ldots$ until

no further counter example can be found. In this implementation, the calls to checking refinements and extracting counterexamples are facilitated by API functions provided by FDR [1].

D. Test case execution

Once all the preceding activities (e.g., intelligence gathering, threat modelling, etc.) are complete, execution of the test cases or test scripts is the next step to be undertaken in the process. As the Figure 2 shows, test generation and test execution are fully automated, they are performed by our software tool. It is worth noting that test case execution is the key element that directly interacts with the target system under test.

V. Testbed Implementation

In this section, we provide implementation details of our testbed, software and hardware components used to construct it, and how it supports our testing approach by carrying out an attack on the automotive OTA using the Uptane reference implementation. The diagram in Figure 3 shows the main components of our testbed.

A. Hardware setup

The image in Figure 4 provides an overview of our proposed testbed. The laptop hosts the server components of the reference implementation of the Uptane framework. Raspberry Pi, the credit-card sized computers, have been used for simulating the primary and secondary ECUs. These computers are equipped with powerful CPUs, various interfaces including LAN and WLAN ports. The primary ECU is attached to the back of a 7-inch touchscreen monitor. A standard network switch has been used for connecting all the devices to each other.

B. Software setup

The code for the Uptane reference implementation has been made available online by its developers at [34]. A guide explaining how to set up and configure the environment to be run on a virtual platform is also available. All the components are assumed to be residing and running on the same environment (i.e., on Linux), with the server being on one console, primary and secondary ECUs on separate consoles. We downloaded the reference implementation code, installed and configured it on each device. Our laptop computer hosting the servers has Ubuntu operating system running on it, whereas both Raspberry Pi computers have the NOOBS operating system.

For a fairly realistic representation of the system, allowing interaction and observation of individual and whole system behaviour, we decided to split the system into three physical tiers, as such servers would run on a laptop and ECUs on separate micro-controllers. Since the reference implementation relies on the TCP/IP for the communication, we used a network switch for interconnecting the server and the ECUs to facilitate communication among them. Figure 4 shows the actual hardware components of our testbed simulating the reference implementation. It is important to note that in

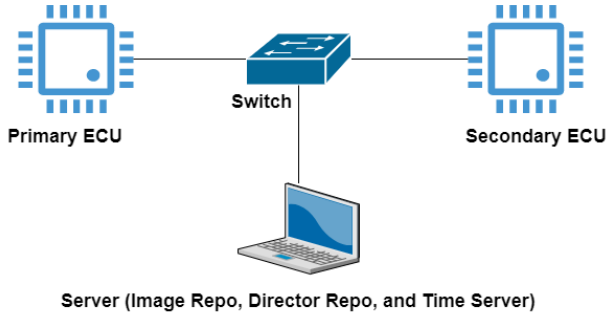

Fig. 3. Schematic diagram of the testbed illustrating the server, the primary and secondary ECUs, and the switch that interconnects them.

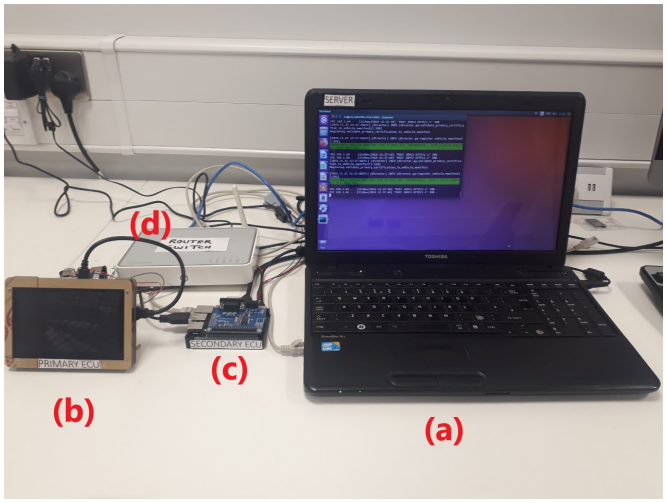

Fig. 4. The testbed for the testing of OTA software update system for automobiles. The setup comprises of (a) a laptop computer (hosting server components for the Uptane framework), (b) the primary ECU, (c) the secondary ECU (both of which implemented on Raspberry Pi 3 micro-controllers), and (d) a switch for interconnecting the components.

the real world, the automotive OTA normally uses mobile communication for delivering updates to the ECUs. However, our simulated setup currently relies entirely on wired medium. This configuration is irrelevant to the simulated attack which is concerned with attacking and compromising Image and Director repositories on the server, rather than targeting the vehicle or its internal components.

To ensure the correct functioning of the reference implementation in a physically distributed environment, essential changes to the configuration files had to be made, which involved IP address configurations of the servers, the primary ECU and the secondary ECU.

VI. Experimentation

There are several types of potential attacks that could be launched against the OTA systems, including eavesdrop attack, replay attack, deny update installation attack, rollback attack, arbitrary software attack and so on. Our simulation attack attempts to compromise both the Image and Director repositories in order to add malicious contents to the firmware images (or new images with malicious contents embedded).

Our threat model assumes that the attacker has gained full access to the OEM repositories (i.e. Image Repository and Director Repository), and has been able to compromise the keys. Following are the details of our attack that we launch by following the four stages of our testing approach (see Figure 2), comprising four different stages.

A. Intelligence gathering in action

This stage is concerned with gathering as much information about the target system as possible by looking into publicly available information, foot-printing, static and dynamic analysis of the code if available and so on. In our case, we have access to the source code and implementation details, and other relevant documentation. After reading publicly available documentation and performing a thorough analysis of the source code along with observation of the system behaviour, we identified several potential threats to the OTA server-side system, one of which is compromising both Image and Director repositories to deploy a malicious firmware image to a target ECU in the vehicle. We chose this particular threat for demonstrating our MBST approach. Since our threat model assumes adversary's unrestricted access to the repository servers, and with the implementation source code availability online, we were able to write some custom code that could be executed from a remote machine to create, sign, and deploy a new firmware image to an ECU in the vehicle.

B. Threat modelling in action

Following this, we then populated our attack tree for a clear and complete understanding of the potential associated actions for the chosen threat. For this study, we chose to experiment and investigate the threat involving compromising a firmware image on the server-side of the OTA system. Using the ADTool, we created and populated our attack tree as displayed in Figure 5. The root node represents the main goal of the attack that is, compromise image. There are two separate subtrees, representing two alternatives to compromising the image file: add new image OR modify an existing image. Both of these trees are SAND (short for Sequential AND) by type, as they both have actions that must be executed in a specific order. Symbolically, it is depicted by an arrow, as can be seen in Figure 5. Both subtrees in our attack tree diagram have four identical nodes and one different node. The actions/sub-goals for the first subtree (on the left) are listed below:

- Add image file - which involves creating a new image file on the Image repository
- Add Image to Repo - this step adds the newly created file to the Image Repo
- Sign image - signs the added image by using the correct signatures
- Add image to Director Repo - the file must be added to the Director repo, after it has been added to and signed by the Image repo.
- Director sign image - this is the last step, which involves signing the image by the Director repo.

The second subtree (on the right) has exactly the same steps except for the first one, which is "make changes" instead of "add new image".

Each of the actions listed above has an associated method that we wrote to execute as an action step. For example, we defined a method for creating an image file, which is responsible for creating a physical image file on the Image repo. Similarly, a method was written for each corresponding action and tied with by entering the name of the method into the description field of the leaf node while creating leaf nodes of the subtree using the ADTool.

C. Test case generation and execution in action

As indicated earlier, the test generation and execution are both fully automated, that is, the tool is capable of automatically generating and executing the test scripts. The output of the preceding stage is an XML (eXtensible Markup Language) file that is then used by our test generation and execution tool. The tool has been programmed to read, interpret and parse the input XML file for identifying individual actions to be carried out. After parsing the attack tree file, the tool generates a list of actions extracted from the leaf nodes of the attack tree along with a list of corresponding method names. As the methods had already been defined, we supplied the code file containing all the methods to the tool as an input, which is subsequently utilised by the software tool to generate an executable test script file as an input to the next step.

To launch the attack, the XML version of the attack-tree (depicted in Figure 5) was supplied to our software tool followed by executing the script file test_generator.py. The tool parses the attack tree and extracts the actions to be carried out as part of the exploitation to compromise the image repositories. As shown in the screenshot in Figure 6, the test script invokes the corresponding custom methods. For example, Action 1 involves creating a new image file for which the method create_image will be executed by the script; similarly, the second action (i.e., Add_Image_to_Image_Repo) invokes the method named add_image_to_imagerepo in the custom-code file. As a result of this action, the newly created firmware image (as shown in Figure 7) file is then added to the Image Repository as depicted in Figure 8. Subsequently, the image file is signed by the Image Repository and copied to and signed by the Director Repository as can be seen in Figure 9.

The primary ECU periodically checks (in our case, we configured the primary and secondary ECUs to look for updates every 60 seconds) for the update and finds that an update is available to download. As shown in Figure 10, the primary ECU successfully downloaded the malicious image file from the server to be supplied to the secondary ECU subsequently.

Finally, the malicious firmware image is downloaded and installed on the secondary ECU as displayed in Figure 11, which proves the attack succeeded by compromising both the repositories (i.e., Image and Director repositories), and consequently the ECU as well. After the successful execution of the test scripts, a report (as shown in Figure 12) is

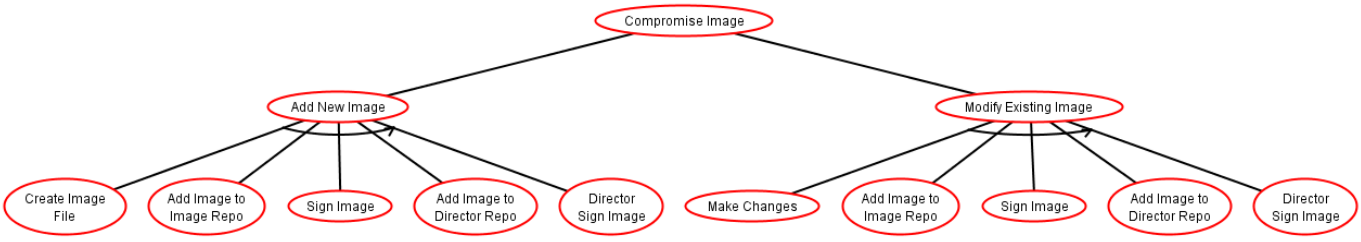

Fig. 5. The attack-tree diagram showing the overall goal (compromise Image), and associated required steps/actions to reach the intended attack goal. There are two possible ways to compromise the firmware image file on the OTA server: by adding a brand new image or by modifying an existing one.

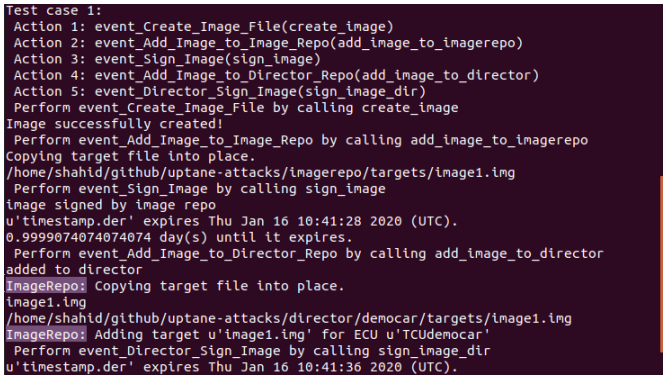

Fig. 6. A screenshot of the test-case execution.

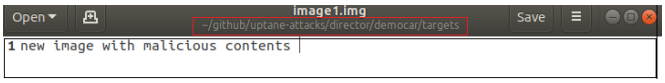

Fig. 7. A screenshot, showing the dummy malicious content of the firmware image file (named "image1.img") created by our script.

generated by the tool summarizing whether the attacks have been successful.

VII. Conclusion

This paper has presented our model-based security testing approach, incorporating an automated software tool for test case generation and execution, and a testbed for cybersecurity evaluation of the automotive OTA. Our approach leverages attack trees for automatic test case generation and execution. Attack trees for threat modelling in our approach help with systematic threat identification and automatic test-case generation, which not only saves time and manual effort, but also helps prevent errors. We used a basic simulated

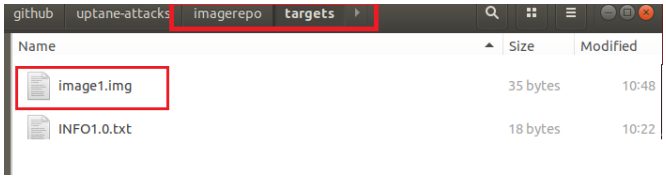

Fig. 8. A screenshot showing the malicious firmware image file (named "image1.img") successfully copied to the Image Repository.

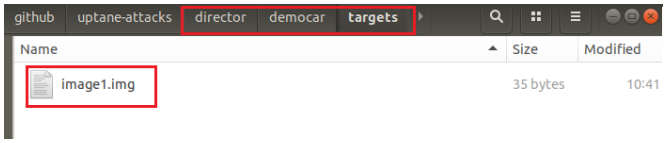

Fig. 9. A screenshot showing the malicious firmware image file (named "image1.img") successfully copied to the Director Repository.

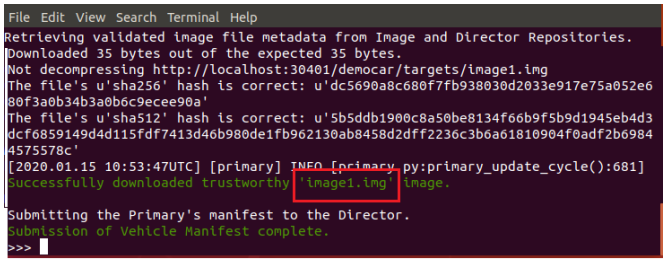

Fig. 10. A screenshot of the Primary ECU, showing the malicious firmware image file (named "image1.img") successfully downloaded from the server.

attack to demonstrate the effectiveness and validity of our testing approach, the software tool, and the testbed. For this purpose, we used a reference implementation of the Uptane. Major contributions of this study include security testing of automotive OTA updates using a systematic model-based approach, automated test-case generation and execution, and

Fig. 11. A screenshot of the Secondary ECU, showing the malicious firmware image file (named "image1.img") successfully downloaded from the Primary ECU and installed.

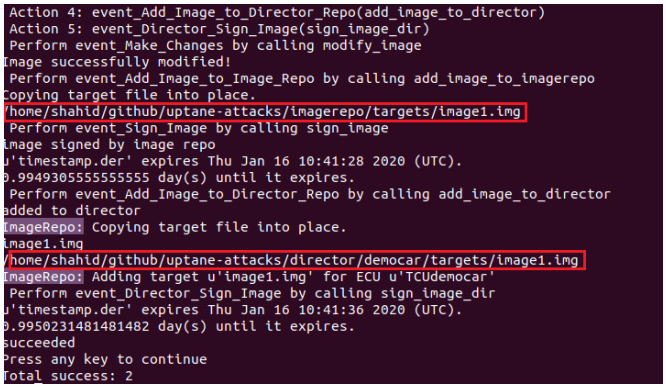

Fig. 12. A screenshot of the test report summarising the test results. It shows that in total two attacks executed and all succeeded.

a cybersecurity testbed. Although, only one type of attack has been demonstrated in this study, more sophisticated attacks can be launched for a more comprehensive evaluation of the OTA updates security. We plan to continue to improve our testing approach, the software tool and the testbed to support our ongoing and future research.

REFERENCES

[1] FDR - CSP Refinement Checker. https://www.cs.ox.ac.uk/projects/fdr.

[2] J. Bryans, L. S. Liew, Sabaliauskaite Nguyen, H. N., S. G., Shaikh, and F. Zhou. Template-based Method for the Generation of Attack Trees. In *2019 WISTP*.

[3] Jeremy Bryans, Hoang Nga Nguyen, and Siraj Shaikh. Attack defense trees with sequential conjunction. In *2019 IEEE HASE*, pages 247–252.

[4] Robert Buttigieg, Mario Farrugia, and Clyde Meli. Security issues in controller area networks in automobiles. In *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 93–98. IEEE, 2017.

[5] Christina Chavez and Carlos Lucena. A metamodel for aspect-oriented modeling. In *Workshop on Aspect-Oriented Modeling with UML (AOSD-2002)*, 2002.

[6] Madeline Cheah, Hoang Nga Nguyen, Jeremy Bryans, and Siraj A. Shaikh. Formalising systematic security evaluations using attack trees for automotive applications. volume 10741, pages 113–129. LNCS. Springer International Publishing, 2018.

[7] Madeline Cheah, Siraj A. Shaikh, Jeremy Bryans, and Paul Wooderson. Building an automotive security assurance case using systematic security evaluations. Computers & Security. 77:360–379.

[8] Madeline Cheah, Siraj A Shaikh, Olivier Haas, and Alastair Ruddle. Towards a systematic security evaluation of the automotive bluetooth interface. *Vehicular Communications*, 9:8–18, 2017.

[9] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, and Harold Ossher. Discussing aspects of AOP. *Communications of the ACM*, 44(10):33–38, 2001.

[10] Christopher E Everett and Damon McCoy. OCTANE (Open Car Testbed and Network Experiments): Bringing cyber-physical security research to researchers and students. In *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*, 2013.

[11] Daniel S. Fowler, Jeremy Bryans, Madeline Cheah, Paul Wooderson, and Siraj A. Shaikh. A method for constructing automotive cybersecurity tests, a CAN fuzz testing example. In *2019 IEEE QRS-C*, pages 1–8.

[12] Daniel S Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, and Paul Wooderson. Fuzz testing for automotive cyber-security. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 239–246. IEEE, 2018.

[13] Daniel S Fowler, Madeline Cheah, Siraj Ahmed Shaikh, and Jeremy Bryans. Towards a testbed for automotive cybersecurity. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 540–541. IEEE, 2017.

[14] Azeem Hafeez, Hafiz Malik, Omid Avatefipour, Prudhvi Raj Rongali, and Shan Zehra. Comparative study of CAN-Bus and FlexRay protocols for in-vehicle communication. Technical report, SAE Tech. Paper, 2017.

[15] John Heneghan, Siraj Ahmed Shaikh, Jeremy Bryans, Madeline Cheah, and Paul Wooderson. Enabling security checking of automotive ECUs with formal CSP models. In *2019 IFIP DSN-W*, pages 90–97. IEEE.

[16] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008.

[17] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. volume 455, pages 339–353. Springer International Publishing.

[18] Ho-Yeon Kim, Young-Hyun Choi, and Tai-Myoung Chung. REES: Malicious software detection framework for MeeGo-In vehicle Infotainment. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*, pages 434–438. IEEE, 2012.

[19] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: security analysis with attack–defense trees. In *Int. conference on quantitative evaluation of systems*, pages 173–176. Springer, 2013.

[20] Trishank Karthik Kuppusamy, Akan Brown, Sebastien Awwad, Damon McCoy, Russ Bielawski, Cameron Mott, Sam Lauzon, André Weimerskirch, and Justin Cappos. Uptane: Securing software updates for automobiles. *14th ESCAR Europe*, 2016.

[21] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. volume 3935, pages 186–198. Springer Berlin Heidelberg.

[22] Hoang Nga Nguyen, Siamak Tavakoli, Siraj Ahmed Shaikh, and Oliver Maynard. Developing a QRNG ECU for automotive security: Experience of testing in the real-world. In *2019 IEEE ICSTW*, pages 61–68.

[23] Phu H Nguyen, Shaukat Ali, and Tao Yue. Model-based security engineering for cyber-physical systems: A systematic mapping study. *Information and Software Technology*, 83:116–135, 2017.

[24] Dennis K Nilsson and Ulf E Larson. Simulated attacks on CAN buses: vehicle virus. In *IASTED International conference on communication systems and networks (AsiaCSN)*, pages 66–72, 2008.

[25] Sidney Nogueira, Augusto Sampaio, and Alexandre Mota. Test generation from state based use case models. *Formal Aspects of Computing*, 26(3):441–490, 2014.

[26] PTES. Penetration testing and execution standard, 2014. http://www.pentest-standard.org/index.php/Main_Page, Last accessed on 2019-12-6.

[27] Caleb Riggs, Carl-Edwin Rigaud, Robert Beard, Tanner Douglas, and Karim Elish. A survey on connected vehicles vulnerabilities and countermeasures. *Journal of Traffic and Logistics Eng. Vol*, 6(1), 2018.

[28] A.W. Roscoe. *Understanding Concurrent Systems*. Springer London, 2010.

[29] Eduardo dos Santos, Andrew Simpson, and Dominik Schoop. A formal model to facilitate security testing in modern automotive systems. *arXiv preprint arXiv:1805.05520*, 2018.

[30] Ina Schieferdecker, Jürgen Großmann, and Martin Schneider. Model-based security testing. *Electronic Proceedings in Theoretical Computer Science*, 80, 02 2012.

[31] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–12. IEEE, 2013.

[32] Tsuyoshi Toyama, Takuya Yoshida, Hisashi Oguma, and Tsutomu Matsumoto. PASTA: Portable automotive security testbed with adaptability.

[33] Uptane Alliance. IEEE-ISTO 6100.1.0.0 uptane standard for design and implementation, n.d. https://uptane.github.io/papers/ieee-isto-6100.1.0.0.uptane-standard.html, Last accessed on 2019-12-6.

[34] Uptane Alliance. Uptane reference implementation code, n.d. https://uptane.github.io/papers/ieee-isto-6100.1.0.0.uptane-standard.html, Last accessed on 2019-12-6.

[35] Xi Zheng, Lei Pan, Hongxu Chen, Rick Di Pietro, and Lynn Batten. A testbed for security analysis of modern vehicle systems. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 1090–1095. IEEE, 2017.

[36] Mohd Azrin Mohd Zulkefli, Pratik Mukherjee, Zongxuan Sun, Jianfeng Zheng, Henry X Liu, and Peter Huang. Hardware-in-the-loop testbed for evaluating connected vehicle applications. *Transportation Research Part C: Emerging Technologies*, 78:50–62, 2017.